

Chapter – 3

String, List, Tuples, Set and Dictionary

String in Python

- A **string** is a sequence of characters enclosed in quotes.
- You can use:
 - a. 'single quotes'
 - b. "double quotes"
 - c. """triple quotes""" for multi-line strings

Example :

```
s = "Hello, Python!"  
print(s)
```

1. Indexing

Each character in a string has an index:

- Left to right (forward): starts from 0
- Right to left (backward): starts from -1

Example :

```
s = "Python"  
print(s[0])      # Output: P  
print(s[5])      # Output: n  
print(s[-1])     # Output: n  
print(s[-6])     # Output: P
```

2. String Operations

a) Concatenation (+)

Combines two or more strings.

Example :

```
a = "Hello"  
b = "World"  
print(a + " " + b)  # Output: Hello World
```

b) Repetition (*)

Repeats the string a given number of times.

Example :

```
word = "Hi"  
print(word * 3)    # Output: HiHiHi
```

c) Membership (in, not in)

Checks if a character or substring exists in the string.

Example :

```
s = "Python"  
print("y" in s)      # Output: True  
print("z" not in s)   # Output: True
```

d) Slicing ([:])

- Extracts parts of a string using the format string[start:end].
- **Note:** The end index is not included.

Example :

```
s = "Python"  
print(s[0:4])    # Output: Pyth  
print(s[2:])     # Output: thon  
print(s[:3])     # Output: Pyt  
print(s[-4:-1])  # Output: tho
```

3. Traversing a String Using Loops

Going through each character in a string one by one.

a) Using for loop:

Example :

```
text = "Hello"  
for ch in text:  
    print(ch)
```

b) Using while loop:

Example :

```
text = "Hello"  
i = 0  
while i < len(text):  
    print(text[i])  
    i += 1
```

4. Built-in String Functions

Python has many useful string functions:

Function	Description	Example
len(s)	Returns length of the string	<code>len("Hello") → 5</code>
s.upper()	Converts to uppercase	<code>"hi".upper() → "HI"</code>
s.lower()	Converts to lowercase	<code>"HI".lower() → "hi"</code>
s.title()	Converts to title case	<code>"hello world".title()</code>
s.strip()	Removes leading/trailing spaces	<code>" hi ".strip() → "hi"</code>
s.replace(a, b)	Replaces a with b	<code>"apple".replace("a", "o")</code>
s.find(x)	Finds index of first occurrence	<code>"hello".find("l") → 2</code>
s.count(x)	Counts occurrences of substring x	<code>"banana".count("a") → 3</code>
s.isalpha()	Checks if all characters are alphabet	<code>"abc".isalpha() → True</code>
s.isdigit()	Checks if all characters are digits	<code>"123".isdigit() → True</code>

Example :

```
s = " Hello World "  
print(len(s))           # Output: 13  
print(s.strip())        # Output: "Hello World"  
print(s.upper())        # Output: " HELLO WORLD "  
print(s.replace("World", "Python")) # Output: " Hello  
                                Python "
```

Summary Table

Feature	What it does	Example
Indexing	Access characters by position	<code>s[0]</code> → first character
Concatenation	Join strings	<code>"Hi" + "There"</code> → <code>"HiThere"</code>
Repetition	Repeat string	<code>"Hi" * 3</code> → <code>"HiHiHi"</code>
Membership	Check for substring	<code>"a" in "cat"</code> → <code>True</code>
Slicing	Extract part of string	<code>s[1:4]</code>
Loop Traversal	Iterate through string	<code>for c in s: print(c)</code>
Functions	Built-in tools for string work	<code>s.upper()</code> , <code>s.find("a")</code>

List in Python

- A list is a collection of ordered, changeable (mutable) items in Python.
- It can contain different data types and is defined using square brackets [].

Example :

```
fruits = ["apple", "banana", "cherry"]  
print(fruits)
```

1) Indexing in List

Like strings, list items can be accessed using indexing:

- Starts from 0 (left to right)
- Negative indexing starts from -1 (right to left)

Example :

```
numbers = [10, 20, 30, 40]  
print(numbers[0])      # Output: 10  
print(numbers[-1])     # Output: 40
```

2) List Operations

a) Concatenation (+)

Joins two lists into one.

Example :

```
a = [1, 2]
b = [3, 4]
print(a + b)    # Output: [1, 2, 3, 4]
```

b) Repetition (*)

Repeats a list multiple times.

Example :

```
print(["Hi"] * 3)    # Output: ['Hi', 'Hi', 'Hi']
```

c) Membership (in, not in)

Checks if an item exists in the list.

Example :

```
colors = ["red", "green", "blue"]
print("red" in colors)    # True
print("yellow" not in colors)    # True
```

d) Slicing

Extracts part of the list using [start:end] (end not included).

Example :

```
marks = [10, 20, 30, 40, 50]
print(marks[1:4])    # Output: [20, 30, 40]
print(marks[:3])    # Output: [10, 20, 30]
print(marks[3:])    # Output: [40, 50]
```

3) Traversing a List

Going through each item in the list.

a) Using for loop:

Example :

```
names = ["Alice", "Bob", "Charlie"]
for name in names:
    print(name)
```

b) Using while loop:

Example :

```
i = 0
while i < len(names):
    print(names[i])
    i += 1
```

4) Built-in List Functions

Function	Description
len(list)	Returns number of elements
list.append(x)	Adds x to end
list.insert(i, x)	Inserts x at index i
list.remove(x)	Removes first occurrence of x
list.pop(i)	Removes item at index i (default last)
list.index(x)	Returns index of first occurrence of x
list.count(x)	Returns count of x in list
list.sort()	Sorts the list
list.reverse()	Reverses the list

Example :

```
a = [3, 1, 2]
a.append(4)
a.sort()
print(a)    # Output: [1, 2, 3, 4]
```

5) Linear Search on List of Numbers

Searches for an element one by one from start to end.

Example :

```
nums = [10, 20, 30, 40]
target = 30
found = False

for num in nums:
    if num == target:
        found = True
        break

print("Found!" if found else "Not Found!")
```

6) Counting the Frequency of Elements in a List

a) Using .count() function:

Example :

```
items = ["apple", "banana", "apple", "cherry", "apple"]
print(items.count("apple")) # Output: 3
```

b) Using manual counting:

Example :

```
nums = [1, 2, 3, 2, 2, 4]
target = 2
count = 0

for n in nums:
    if n == target:
        count += 1

print("Frequency:", count) # Output: 3
```

Summary

Concept	Description
List	Ordered collection, mutable
Indexing	Access items using []
Concatenation / Repetition	Combine / repeat lists
Membership	Check presence using in / not in
Slicing	Get sublists using [start:end]
Traversal	Use loops to access all items
Built-in Functions	append(), count(), sort(), etc.
Linear Search	Check if a number exists by going one-by-one
Frequency Count	Count how many times an element appears

Tuples in Python

- Tuples are ordered, immutable collections of items.
- They can store elements of different data types, and once created, their values cannot be changed (unlike lists).

1. Creating Tuples

You can create tuples by placing items inside parentheses () separated by commas.

Example :

```
# Creating an empty tuple
empty_tuple = ()
```



```
# Tuple with elements
my_tuple = (10, 20, 30)

# Tuple without parentheses (tuple packing)
another_tuple = 1, 2, 3

# Tuple with mixed data types
mixed_tuple = (1, "hello", 3.14)
```

2. Initializing Tuples

You can initialize a tuple directly during declaration.

Example :

```
names = ("Alice", "Bob", "Charlie")
one_element = (5,)
```

3. Accessing Elements

Tuples are ordered, so elements can be accessed using indexing and slicing.

Example :

```
t = (10, 20, 30, 40, 50)
# Accessing by index
print(t[0])      # Output: 10
print(t[-1])     # Output: 50
# Slicing
print(t[1:4])    # Output: (20, 30, 40))
```

4. Tuple Assignment (Unpacking)

You can unpack tuples into variables.

Example :

```
point = (5, 10)
x, y = point
print(x)    # Output: 5
print(y)    # Output: 10
```

5. Operations on Tuples

Tuples support basic operations like concatenation, repetition, and membership tests.

Example :

```
t1 = (1, 2, 3)
t2 = (4, 5)
# Concatenation
t3 = t1 + t2      # (1, 2, 3, 4, 5)
# Repetition
t4 = t1 * 2        # (1, 2, 3, 1, 2, 3)
# Membership
print(2 in t1)     # True
```

6. Tuple Methods

Tuples support only two built-in methods:

Example :

```
t = (1, 2, 2, 3, 4)
# count(x): number of times x appears
print(t.count(2))  # Output: 2
# index(x): first index of x
print(t.index(3))  # Output: 3
```

7. Built-in Functions for Tuples

Some useful built-in functions that can be used with tuples:

Example :

```
numbers = (10, 20, 30)

print(len(numbers))      # Number of elements: 3
print(max(numbers))      # Maximum value: 30
print(min(numbers))      # Minimum value: 10
print(sum(numbers))      # Sum of elements: 60
```

8. Nested Tuples

Tuples can contain other tuples or collections as elements.

Example :

```
nested = ((1, 2), (3, 4), (5, 6))
# Accessing nested elements
print(nested[0])           # (1, 2)
print(nested[0][1])        # 2
```

Summary

Feature	Tuple
Mutable	✗ No
Indexed	✓ Yes
Allows duplicates	✓ Yes
Methods available	.count(), .index()
Useful for	Fixed data, unpacking, safe data passing

Set in Python

A set is a built-in Python data type that:

- Stores multiple items in a single variable.
- Is unordered (no fixed sequence).
- Has no duplicate values.
- Is mutable (you can add or remove elements), but the elements themselves must be immutable (numbers, strings, tuples).

1. Creating a Set

You can create a set in multiple ways:

Example :

```
# Using curly braces {}
fruits = {"apple", "banana", "cherry"}

# Using set() constructor
numbers = set([1, 2, 3, 4])

# Empty set
empty_set = set()    # NOT {} because {} creates an empty
                     # dictionary
```

Note: Sets don't allow duplicates — if you try, they're automatically removed.

Example :

```
duplicates = {1, 2, 2, 3}
print(duplicates)    # Output: {1, 2, 3}
```

2. Traversing a Set

- Since sets are unordered, you can't access elements by index.
- You use a for loop:

Example :

```
colors = {"red", "green", "blue"}
for color in colors:
    print(color)
```

3. Adding Data to a Set

Example :

```
my_set = {1, 2, 3}
my_set.add(4)          # Adds one element
print(my_set)         # {1, 2, 3, 4}

my_set.update([5, 6])  # Adds multiple elements at once
print(my_set)         # {1, 2, 3, 4, 5, 6}
```

4. Removing Data from a Set\

Example :

```
my_set = {1, 2, 3, 4}
my_set.remove(3)      # Removes element, throws error if not
                        # found
print(my_set)         # {1, 2, 4}
my_set.discard(5)     # Removes element if present, no error
                        # if missing
print(my_set)         # {1, 2, 4}
popped_item = my_set.pop() # Removes and returns a random
                        # element

print(popped_item)
print(my_set)
my_set.clear()        # Removes all elements
print(my_set)         # set()
```

5. Performing Set Operations

Let's take two example sets:

Example :

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
```

a) Union (| or .union())

Combines all elements from both sets (no duplicates).

Example :

```
print(A | B)          # {1, 2, 3, 4, 5, 6}
print(A.union(B))     # Same result
```

b) Intersection (& or .intersection())

Elements common to both sets.

Example :

```
print(A & B)          # {3, 4}
print(A.intersection(B)) # Same result
```

c) Difference (- or .difference())

Elements in first set but **not** in the second.

Example :

```
print(A - B)           # {1, 2}
print(A.difference(B)) # Same result
```

d) Symmetric Difference (^ or .symmetric_difference())

Elements in either set but **not** both.

Example :

```
print(A ^ B)           # {1, 2, 5, 6}
print(A.symmetric_difference(B)) # Same result
```

Summary Table:

Operation	Symbol	Method Name
Union	,	,
Intersection	&	.intersection()
Difference	-	.difference()
Symmetric Difference	^	.symmetric_difference()

Dictionary in Python

A dictionary is a key-value pair data structure in Python.

- Keys are unique and immutable (strings, numbers, tuples).
- Values can be any data type and can repeat.
- Dictionaries are unordered in Python versions < 3.7 and insertion-ordered from Python 3.7 onward.
- Dictionaries are mutable — we can add, modify, and remove items.

Example :

```
student = {  
    "name": "Alice",  
    "age": 20,  
    "course": "Computer Science"  
}
```

1. Accessing Items in a Dictionary Using Keys

You access a value by referring to its key.

Example :

```
print(student["name"])    # Alice  
print(student["age"])    # 20
```

Note :

- If the key doesn't exist, [] will raise a KeyError.
- To avoid errors, use .get():

Example :

```
print(student.get("course"))    # Computer Science  
print(student.get("marks", 0))  # 0 (default value)
```

2. Mutability of Dictionary

Since dictionaries are mutable, you can change them after creation.

a) Adding a New Item

Example :

```
student["marks"] = 95
print(student)
```

b) Modifying an Existing Item

Example :

```
student["age"] = 21
print(student)
```

3. Built-in Dictionary Functions and Methods

Method / Function	Purpose
len(dict)	Returns number of key-value pairs
dict.keys()	Returns all keys
dict.values()	Returns all values
dict.items()	Returns all key-value pairs as tuples
dict.get(key, default)	Returns value for key or default if not found
dict.update(other_dict)	Adds/updates multiple key-value pairs
dict.pop(key)	Removes the item with given key
dict.popitem()	Removes and returns last inserted item
dict.clear()	Removes all items
dict.copy()	Returns a shallow copy

Example :

```
# Length of dictionary
print(len(student))  # 4
# Keys, Values, Items
print(student.keys())
# dict_keys(['name', 'age', 'course', 'marks'])
print(student.values())
# dict_values(['Alice', 21, 'Computer Science', 95])
print(student.items())
# dict_items([('name', 'Alice'), ('age', 21), ...])

# Update multiple values
student.update({"marks": 98, "city": "Delhi"})
print(student)
# {'name': 'Alice', 'age': 21, 'course': 'Computer
Science', 'marks': 98, 'city': 'Delhi'}

# Remove an item
student.pop("age")
print(student)
```